2020 자율주행 교육

WeGo 위고 주식회사

목차

1. CNN
2. 대표적인 네트워크
3. Darknet & Yolo

WeGo

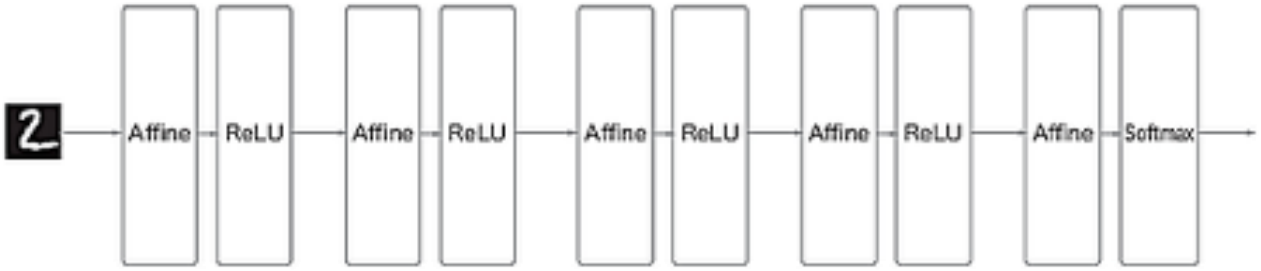
01

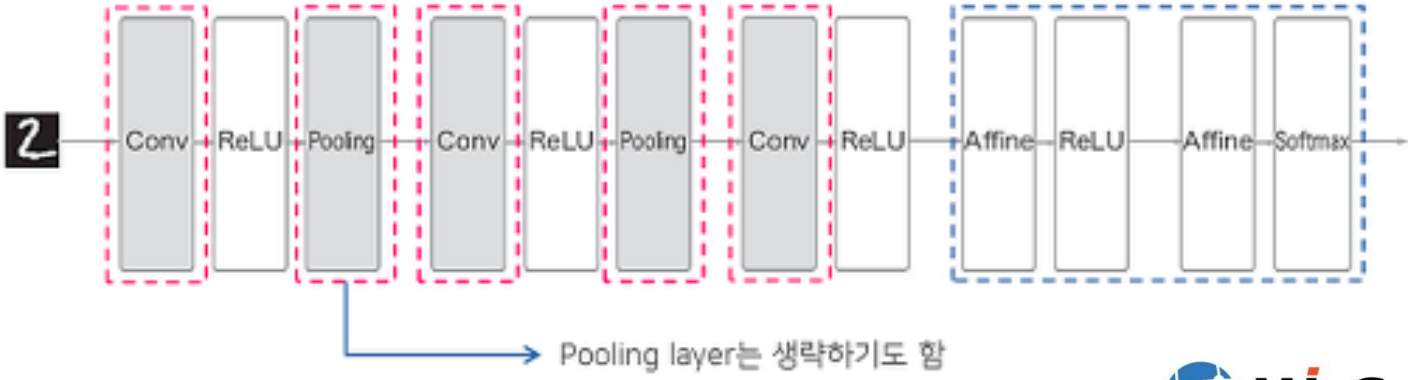
CNN

WeGo

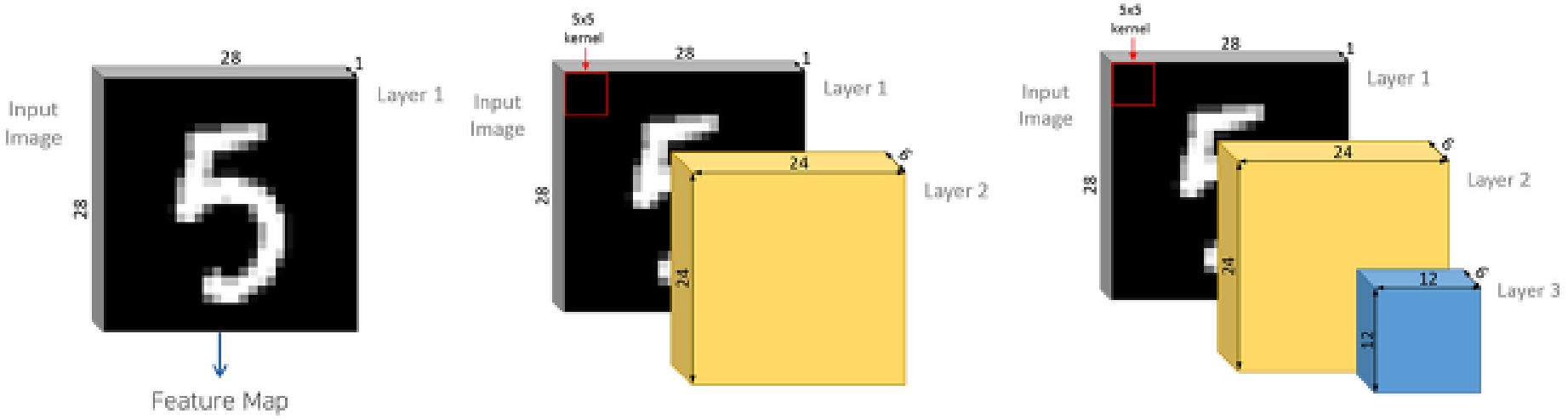
01. CNN

- 기존 방법의 경우 완전 연결(fully-connected)된 계층(Affine 계층)을 이용하여 구현
- CNN은 Convolution 계층과 Pooling 계층을 추가된 형태
- 마지막은 일반적인 Fully-connected와 Softmax를 조합하여 그대로 사용

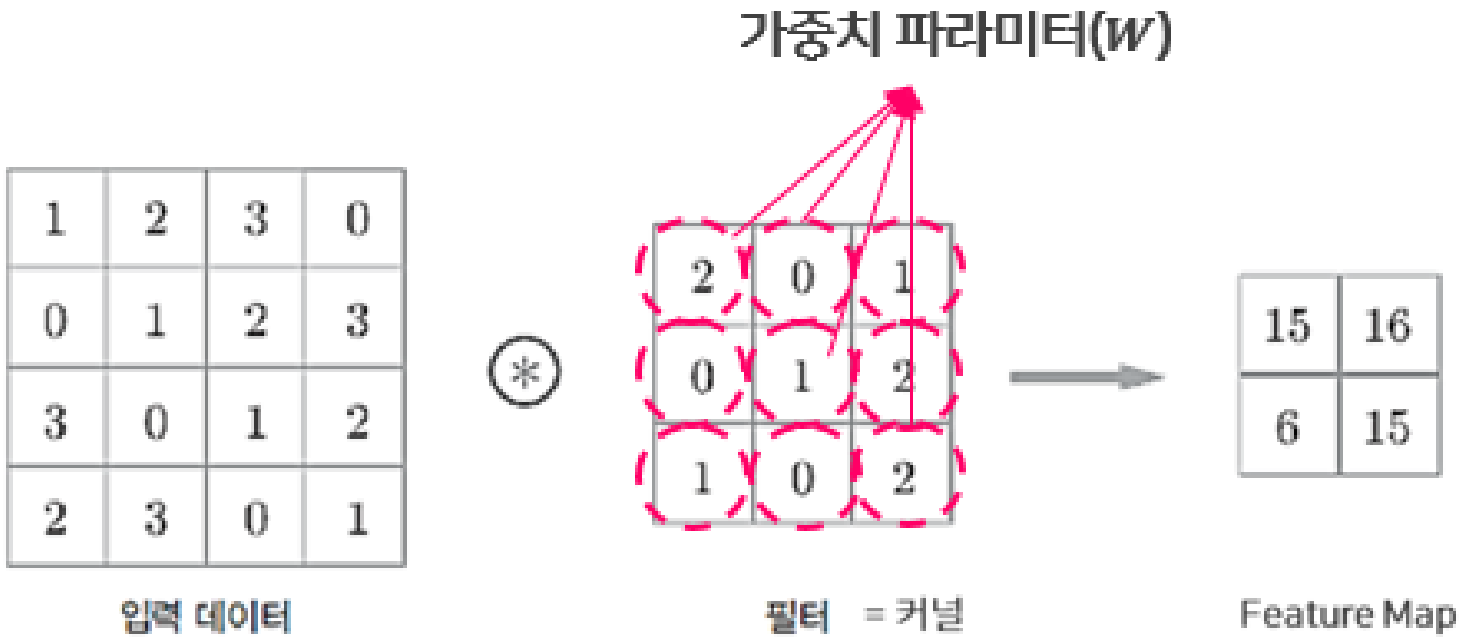




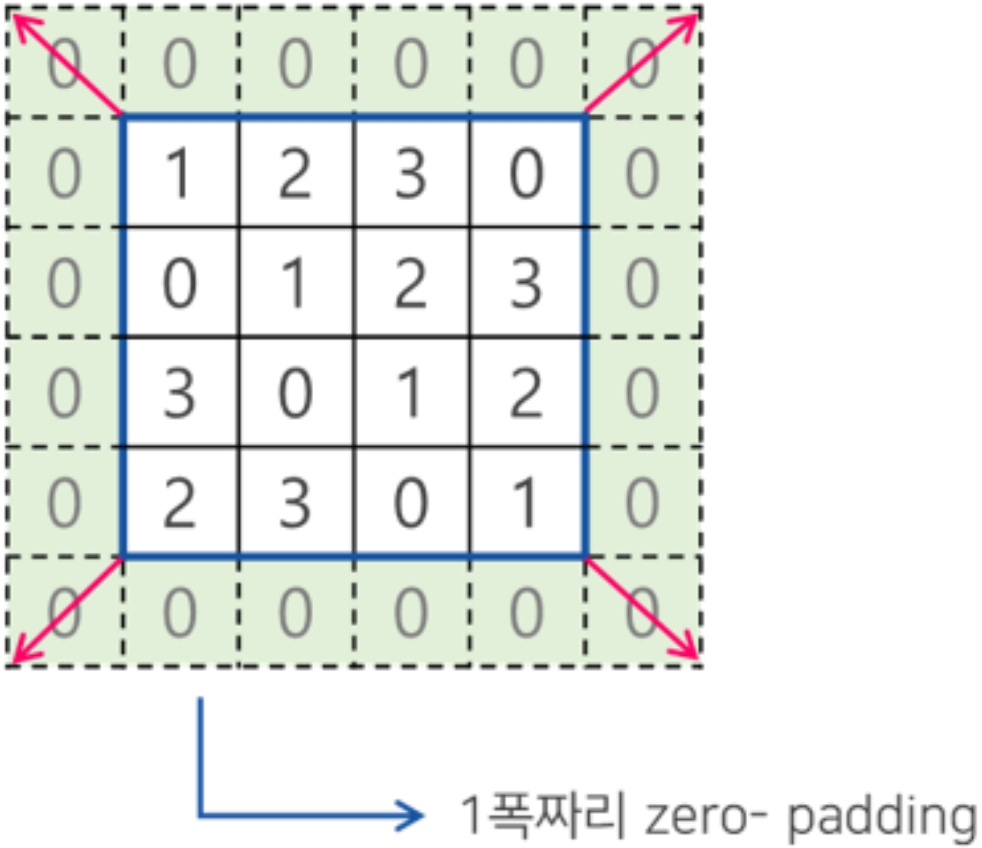
- Affine 계층에서는 모든 형상을 1차원화해야만 정상적인 사용이 가능
- 반면 Convolution 계층에서는 각 계층 사이에, 입체적인 데이터가 전달됨



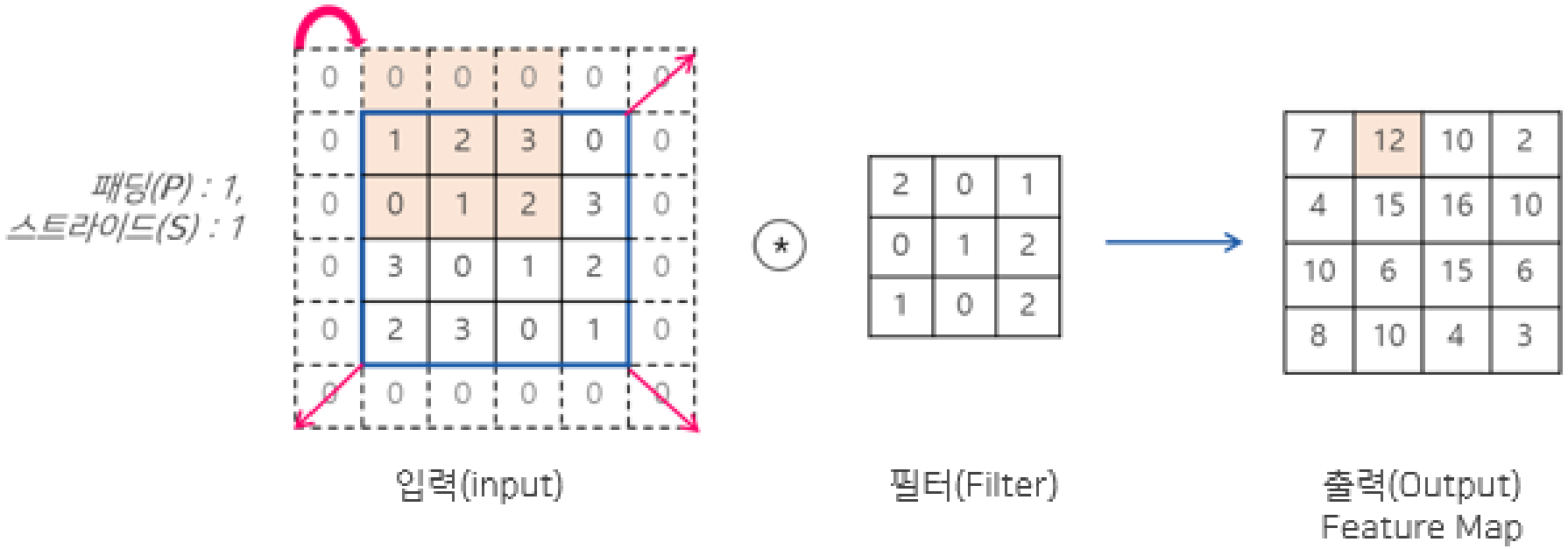
- Convolution 연산 - 이미지 필터 연산



- Padding - Convolution 수행 전, 출력 데이터의 크기를 조절하기 위해, 주변 값을 채우는 것을 의미, 일반적으로 0으로 채우는 zero-padding을 사용



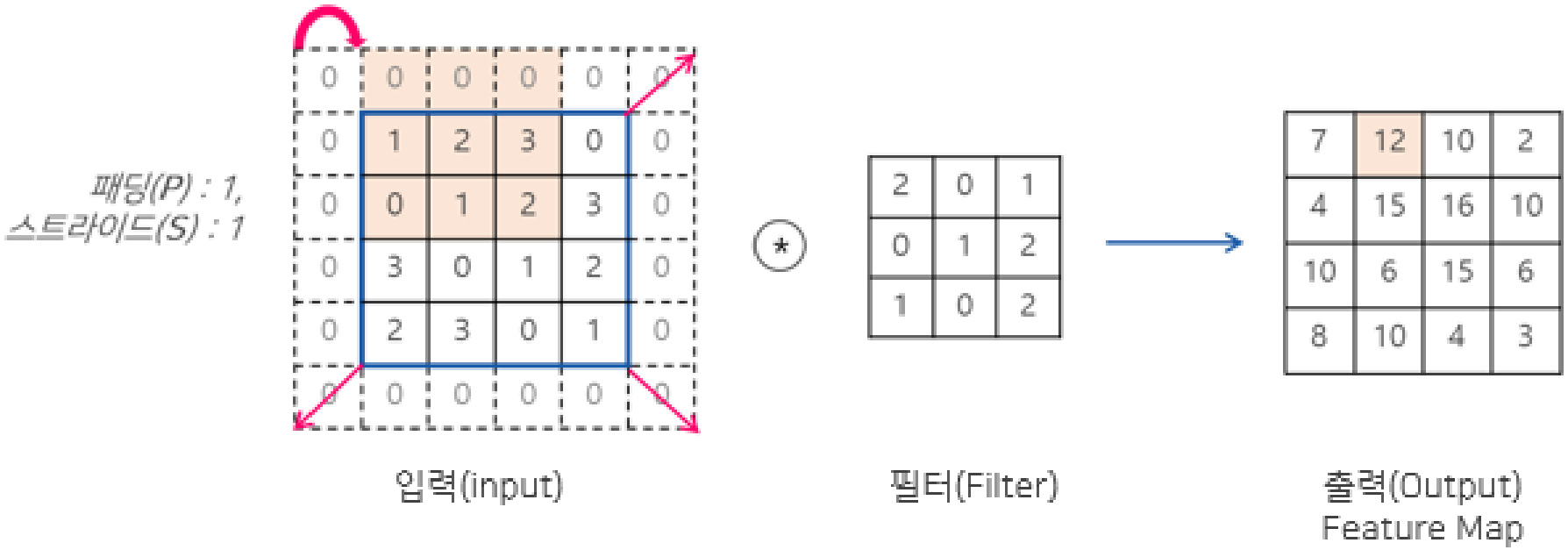
- Stride - 필터가 적용될 때, 이동할 간격, 마찬가지로 출력의 크기를 조절하기 위해서 사용하며, 기본 값은 1로 적용



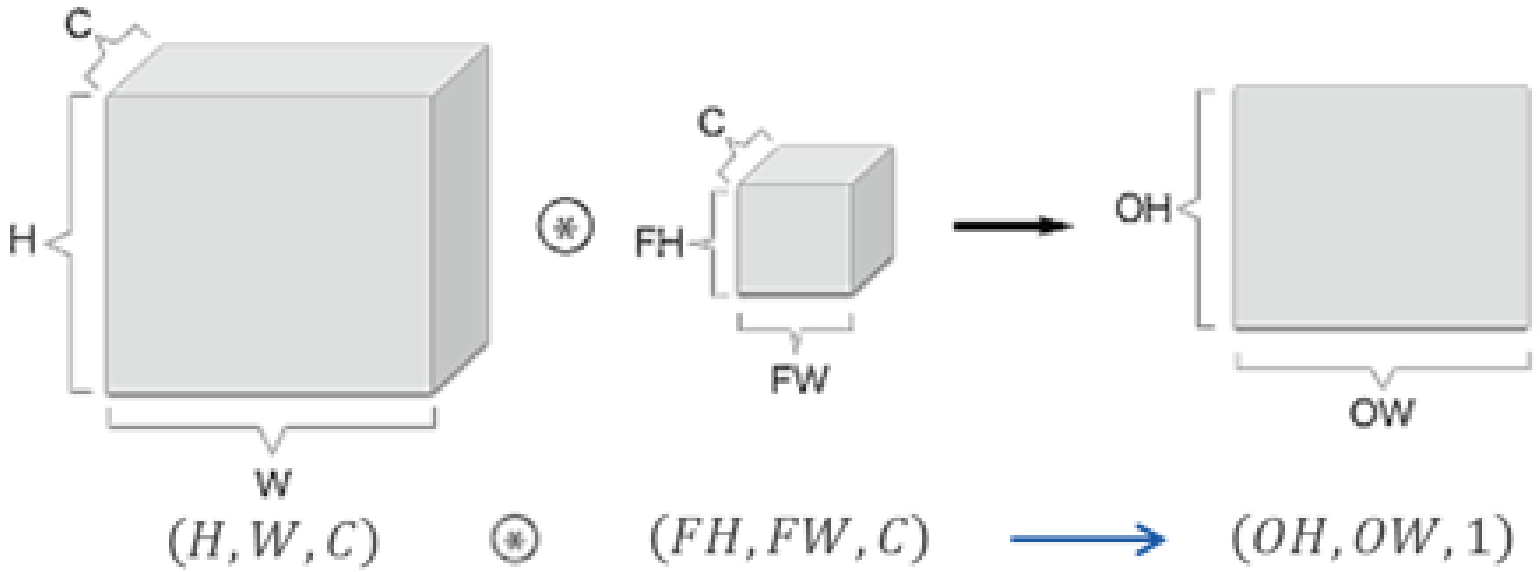
- 최종 출력의 크기 계산

$$(OH, OW) = (\frac{IH + 2P - FH}{S} + 1, \frac{W + 2P - FW}{S} + 1)$$

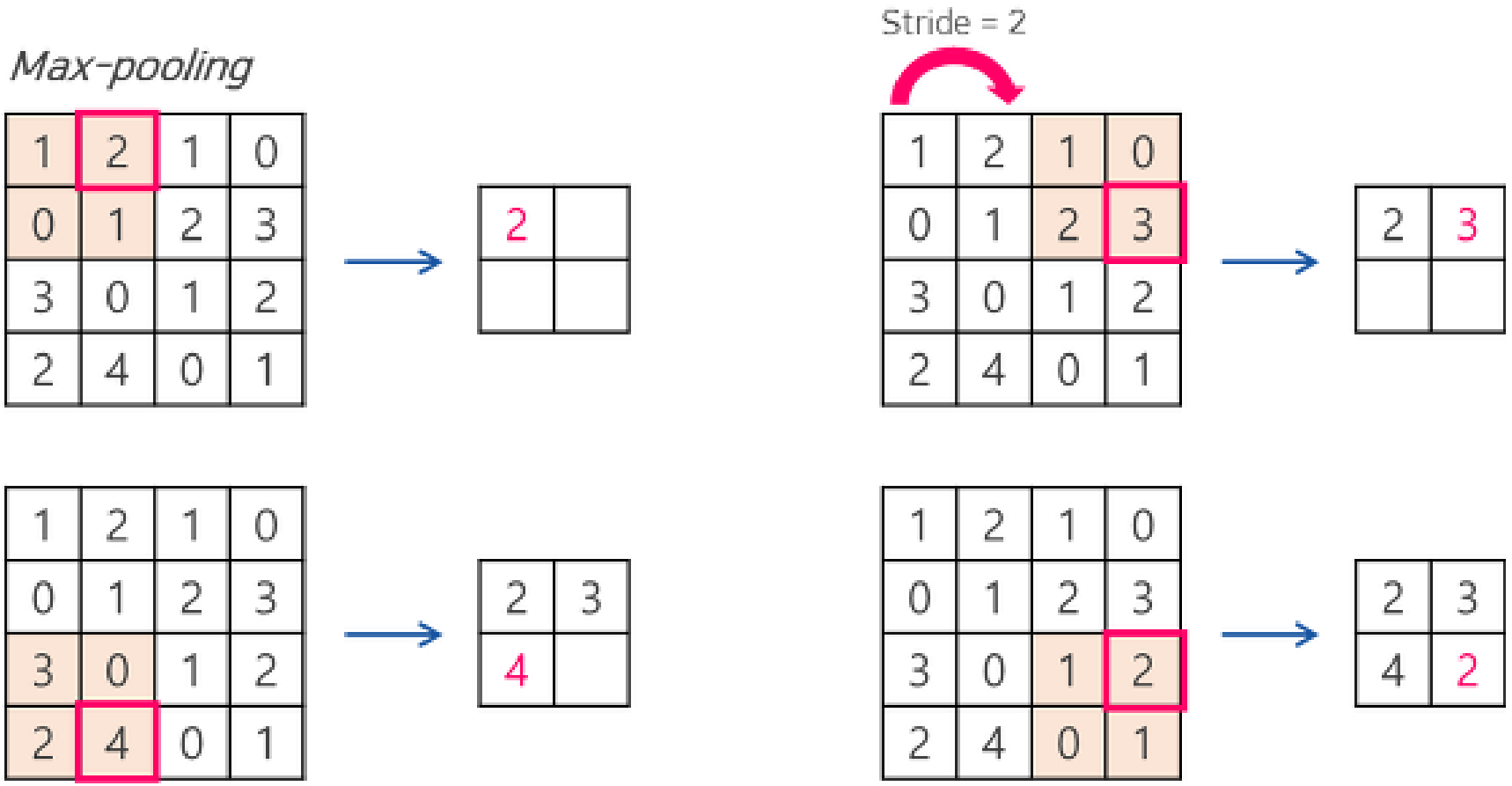
$$(OH, OW) = Output\ size, (IH, IW) = Input\ size, (FH, FW) = Filter\ size, P = padding, S = Stride$$



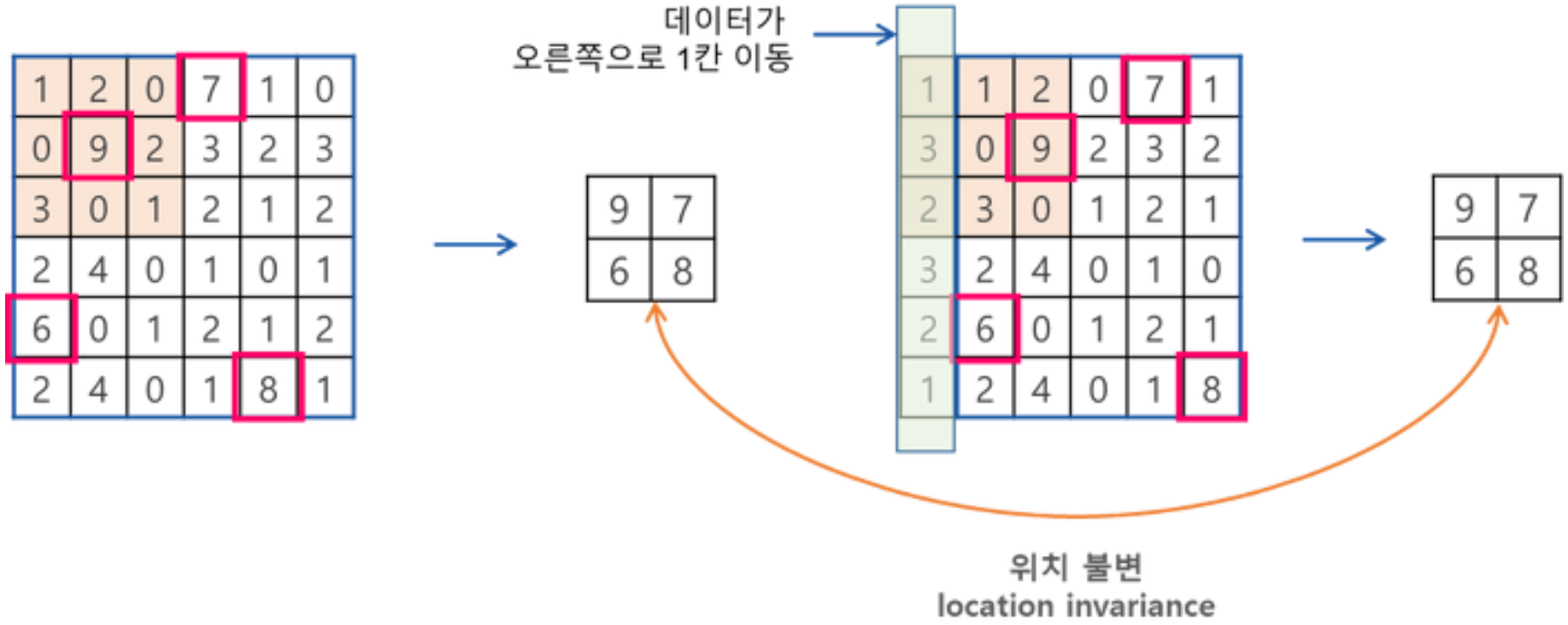
- 3차원 데이터의 합성곱 연산 - 입력과 필터의 채널 수가 동일해야 함



- 풀링 계층



- 풀링 계층의 경우, 매개변수가 없으며, 입력 채널과 동일하게 출력 채널 수가 유지
- 입력 데이터가 약간 변하는 것에 대해서는 Robust한 결과를 출력할 수 있음



- 이미지를 평탄화 시키는 함수

```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    N, C, H, W = input_data.shape
    out_h = (H + 2*pad - filter_h)//stride + 1
    out_w = (W + 2*pad - filter_w)//stride + 1

    img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride*out_h
        for x in range(filter_w):
            x_max = x + stride*out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N*out_h*out_w, -1)
    return col
```

```
class Convolution:
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

    def forward(self, x):
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        out_h = int(1 + (H + 2*self.pad – FH) / self.stride)
        out_w = int(1 + (W + 2*self.pad – FW) / self.stride)

        col = im2col(x, FH, FW, self.stride, self.pad)
        col_W = self.W.reshape(FN, -1).T
        out = np.dot(col, col_W) + self.b

        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        return out
```

```
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - 2*self.pad – FH) / self.stride)
        out_w = int(1 + (W - 2*self.pad – FW) / self.stride)

        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)
        out = np.max(col, axis=1)

        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        return out
```

```
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - 2*self.pad – FH) / self.stride)
        out_w = int(1 + (W - 2*self.pad – FW) / self.stride)

        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        col = col.reshape(-1, self.pool_h*self.pool_w)
        out = np.max(col, axis=1)

        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        return out
```

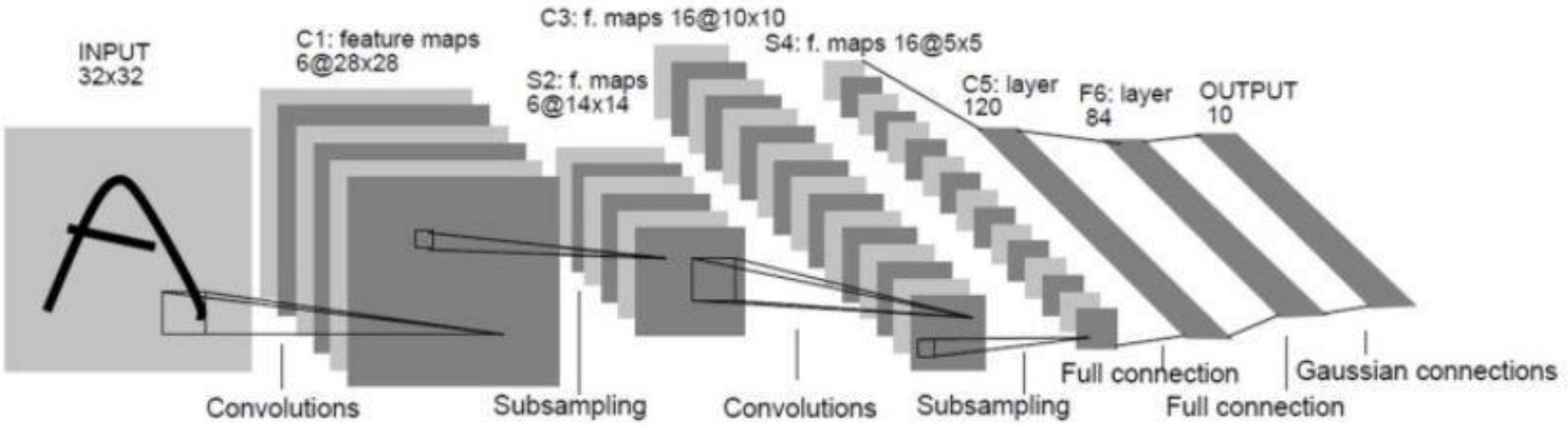
02

대표적인 네트워크

WeGo

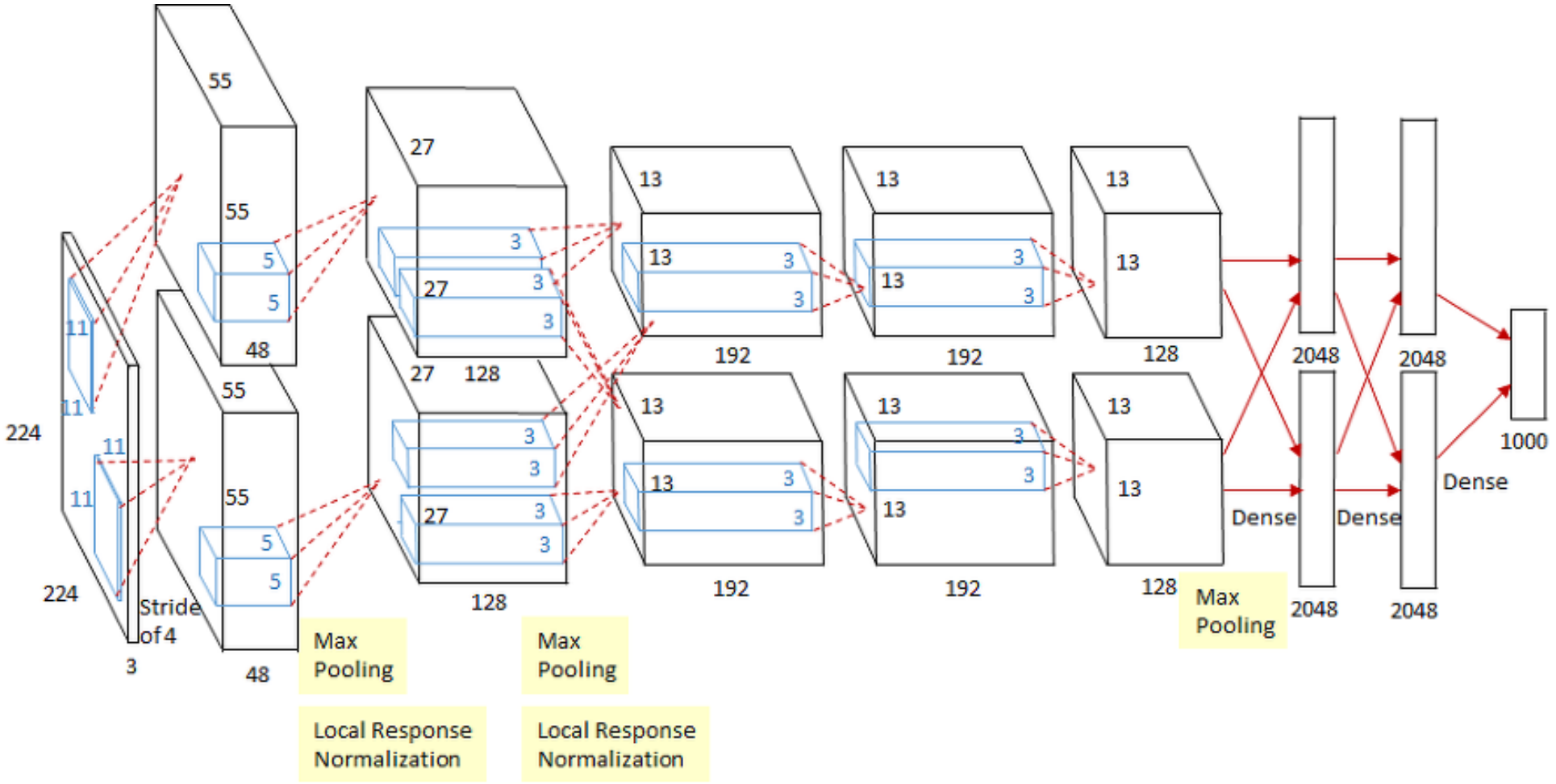
02. 대표적인 네트워크

- LeNet(1998) - CNN의 원조
- 손글씨 숫자 인식 네트워크 – Subsampling반복 후, Fully connected layer이용

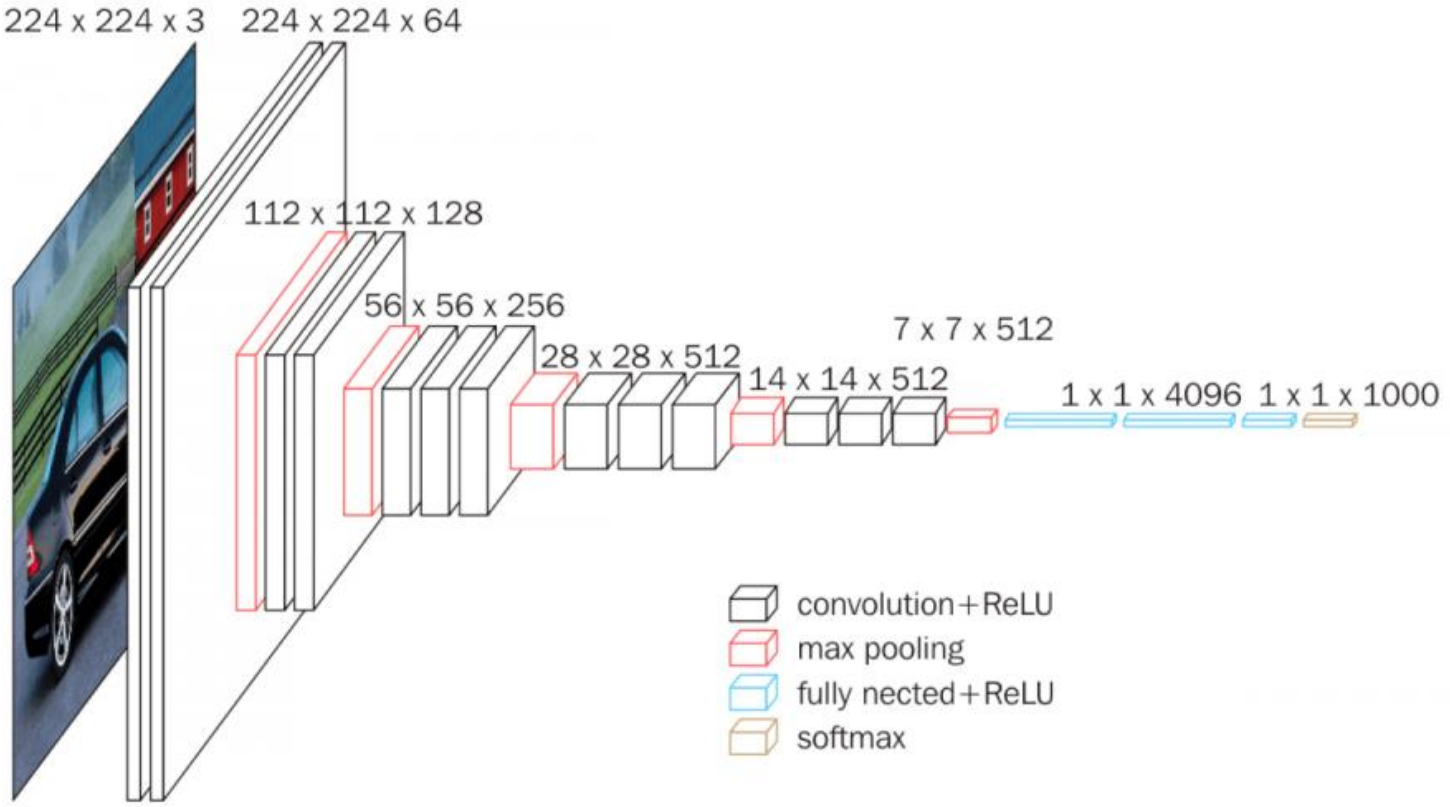


02. 대표적인 네트워크

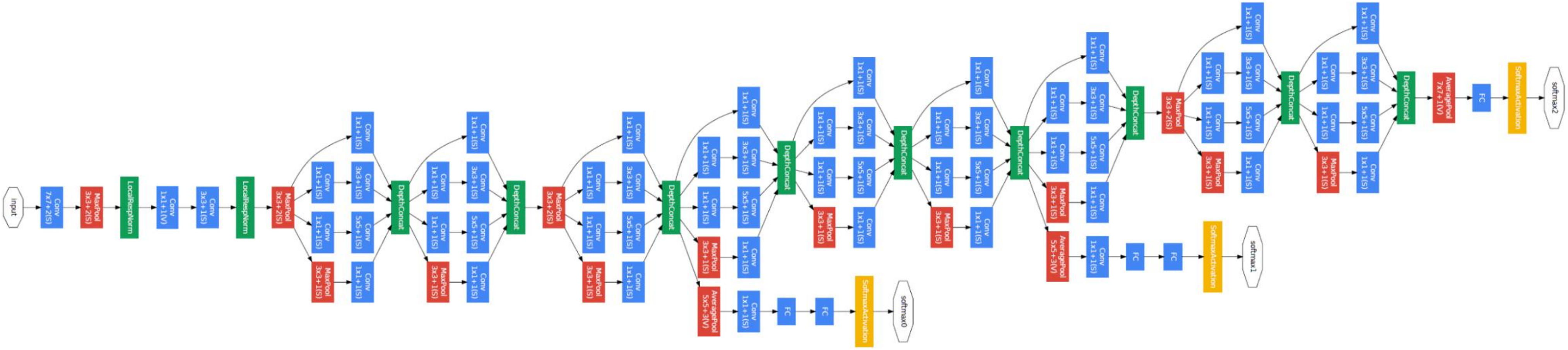
- AlexNet(2012)
- 2개의 GPU로 병렬연산을 위해 병렬 구조로 설계



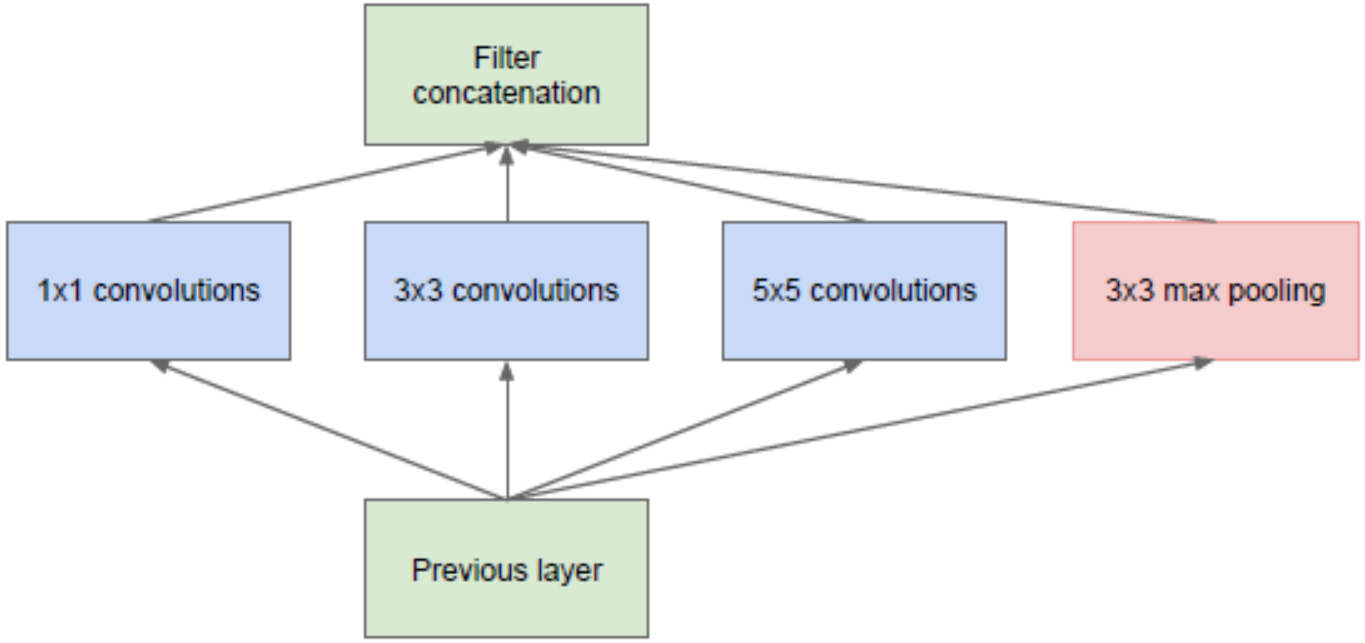
- VGG
- 구조가 간단하고, 변형하여 사용하기가 쉬움



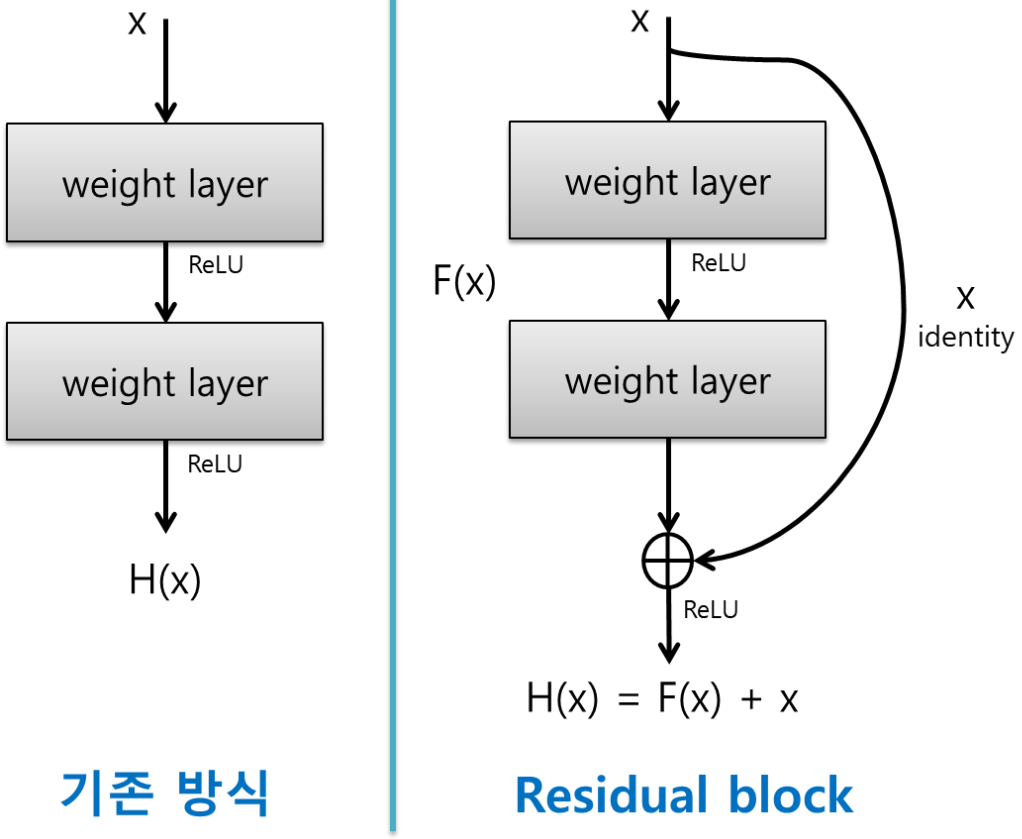
- GoogLeNet



- GoogLeNet
- 인셉션 구조 - 1 x 1 Convolution을 이용하여, 특성맵의 개수를 줄이고 속도 향상



- ResNet
- F(x) + x를 최소화 하기 위해, F(x)를 0에 가깝게 만드는 것이 목적



03

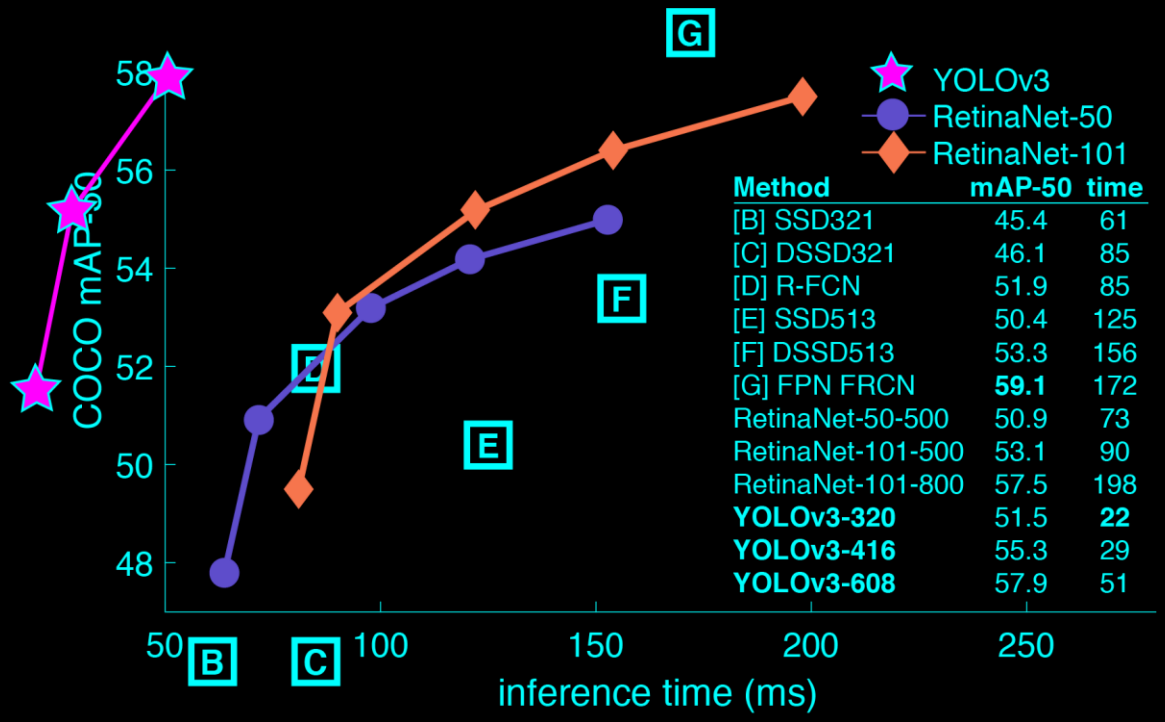
Darknet & Yolo

WeGo

03. Darknet & Yolo

03. Darknet & Yolo

- Darknet - 딥러닝을 위한 Framework
- Yolo - You only look once의 줄임말고, real-time object detection을 위한 시스템
- Pascal Titan X 기준으로 30FPS, mean Average Precision(mAP) 가 57.9%



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

- Yolo Network 구조

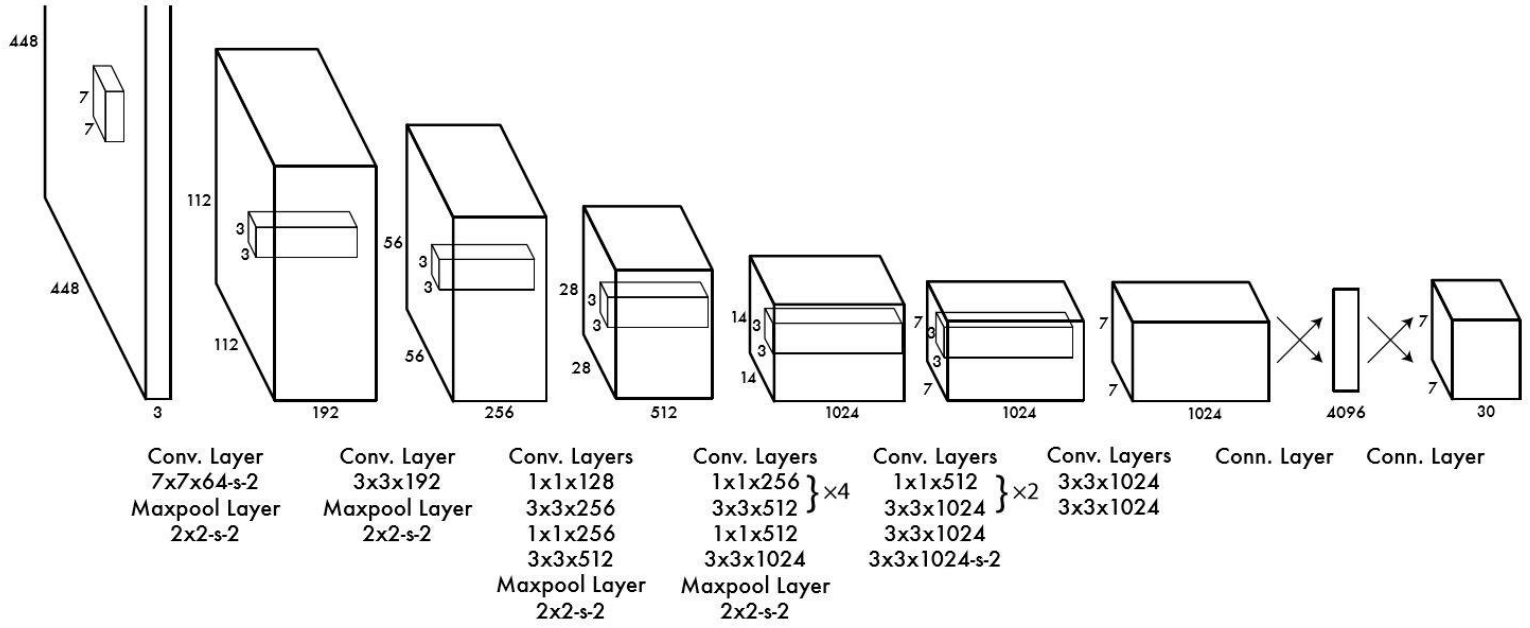


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

WeGo